Intellectual Output_O1:
EMERALD e-book for developing of biomimetic mechatronic systems

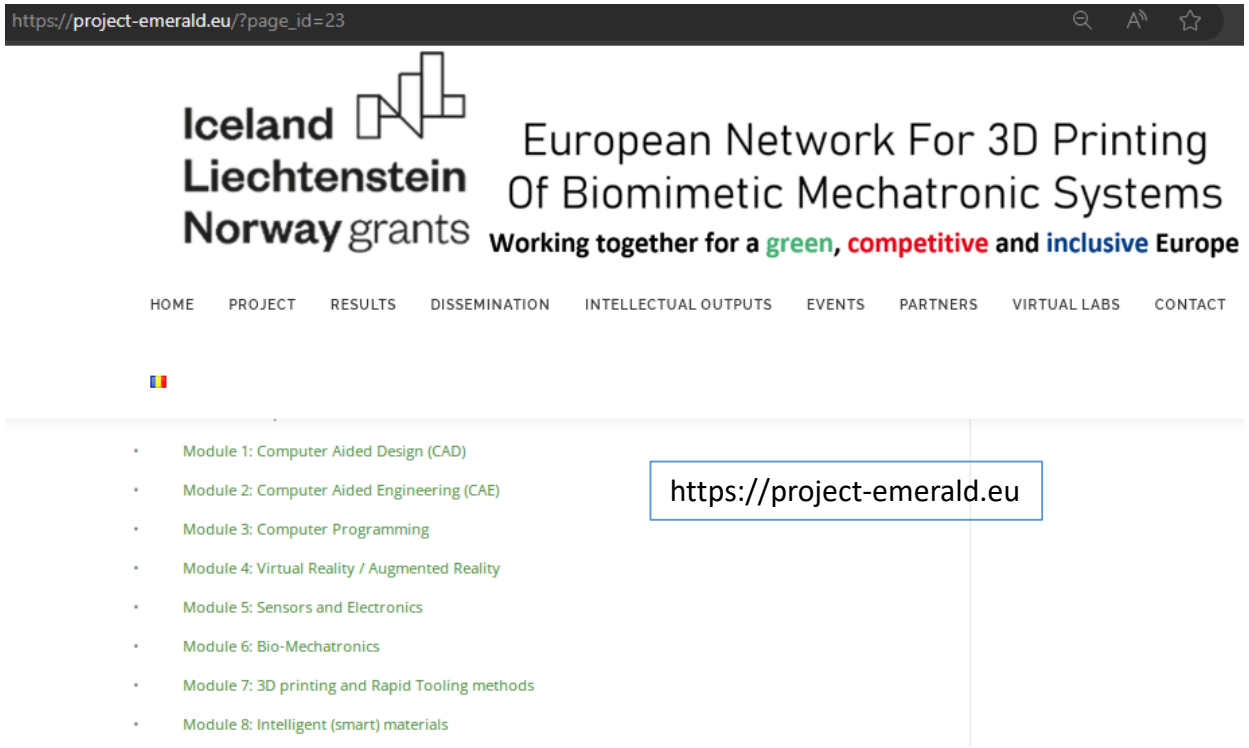
MODULE 3
Computer Programming

Martin Zelenay, Michal Gallia

BIZZCOM

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

EUROPEAN NETWORK FOR 3D PRINTING OF BIOMIMETIC MECHATRONIC SYSTEMS - EMERALD



https://project-emerald.eu

EUROPEAN NETWORK FOR 3D PRINTING OF BIOMIMETIC MECHATRONIC SYSTEMS

MODULE 3 – Computer Programming

Project Title	**European network for 3D printing of biomimetic mechatronic systems 21-COP-0019**
Output	**IO1 - EMERALD e-book for developing of biomimetic mechatronic systems**
Module	**Module 3 – Computer Programming**
Date of Delivery	**July 2022**
Authors	**Martin Zelenay, Michal Gallia**
Version	**1.1.**

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

EUROPEAN NETWORK FOR 3D PRINTING OF BIOMIMETIC MECHATRONIC SYSTEMS - EMERALD

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

Python is free and open source, available for all operating systems from python.org. In particular we want a Python install where you can do two things:

- Run an existing python program, such as hello.py
- Run the Python interpreter interactively, so you can type code right at it.

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

As a first step, download the google-python-exercises.zip file and unzip it someplace where you can work on it. The resulting google-python-exercises directory contains many different python code exercises you can work on. In particular, google-python-exercises contains a simple hello.py file you can use in the next step to check that Python is working on your machine.

Python on Linux, Mac OS X, and other OS

```
~/google-python-exercises$ python hello.py
Hello World
~/google-python-exercises$ python hello.py Alice
Hello Alice
```

If python is not installed, see the Python.org download page. To run the Python interpreter interactively, just type `python` in the terminal:

```
~/google-python-exercises$ python3
Python 3.X.X (XXX, XXX XX XXXX, 03:41:42) [XXX] on XXX
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 1
2
>>> you can type expressions here.. use ctrl-d to exit
```

Python on Windows

Windows Notepad++ -- Tabs: Settings > Preferences > Edit Components > Tab settings, and Settings > Preferences > MISC for auto-indent. Line endings: Format > Convert, set to Unix.

JEdit (any OS) -- Line endings: Little 'U' 'W' 'M' on status bar, set it to 'U' (for Unix line-endings).

Windows Notepad or Wordpad -- do not use.

Mac BBEdit -- Tabs: At the top, BBEdit > Preferences (or Cmd + , shortcut). Go to Editor Defaults section and make sure Auto-indent and Auto-expand tabs are checked. Line endings: In Preferences go to Text Files section and make sure Unix (LF) is selected under Line breaks.

Mac TextEdit -- do not use.

Unix pico -- Tabs: Esc-q toggles tab mode, Esc-i to turns on auto-indent mode.

Unix emacs -- Tabs: manually set tabs-inserts-spaces mode: `M-x set-variable (return) indent-tabs-mode (return) nil`.

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

Python Introduction

Language Introduction

```
   $ python        ## Run the Python interpreter
Python 3.X.X (XXX, XXX XX XXXX, 03:41:42) [XXX] on XXX
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 6       ## set a variable in this interpreter session
>>> a           ## entering an expression prints its value
6
>>> a + 2
8
>>> a = 'hi'    ## 'a' can hold a string just as well
>>> a
'hi'
>>> len(a)      ## call the len() function on a string
2
>>> a + len(a)  ## try something that doesn't work
Traceback (most recent call last):
  File "", line 1, in
TypeError: can only concatenate str (not "int") to str
>>> a + str(len(a))  ## probably what you really wanted
'hi2'
>>> foo         ## try something else that doesn't work
Traceback (most recent call last):
  File "", line 1, in

NameError: name 'foo' is not defined
>>> ^D          ## type CTRL-d to exit (CTRL-z in Windows/DOS terminal)
```

Python source code

Here's a very simple `hello.py` program (notice that blocks of code are delimited strictly using indentation rather than curly braces — more on this later!):

```
#!/usr/bin/env python

# import modules used here -- sys is a very standard one
import sys

# Gather our code in a main() function
def main():
    print('Hello there', sys.argv[1])
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

# Standard boilerplate to call the main() function to begin
# the program.

if __name__ == '__main__':
    main()
```

Running this program from the command line looks like:

```
$ python hello.py Guido

Hello there Guido

$ ./hello.py Alice  ## without needing 'python' first (Unix)

Hello there Alice
```

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

Imports, Command-line arguments, and len()

User-defined Functions

Functions in Python are defined like this:

```
# Defines a "repeat" function that takes 2 arguments.
def repeat(s, exclaim):
    """
    Returns the string 's' repeated 3 times.
    If exclaim is true, add exclamation marks.
    """

 result = s + s + s # can also use "s * 3" which is faster (Why?)
 if exclaim:
     result = result + '!!!'
 return result
```

Code Checked at Runtime

```
def main():
 if name == 'Guido':
     print(repeeeet(name) + '!!!')
 else:
     print(repeat(name))
```

More on Modules and their Namespaces

Commonly used modules/packages include:

sys — access to exit(), argv, stdin, stdout, ...

re — regular expressions

os — operating system interface, file system

You can find the documentation of all the Standard Library modules and packages at http://docs.python.org/library.

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

Python Strings

```
    s = 'hi'
  print s[1]          ## i
  print len(s)        ## 2
  print s + ' there'  ## hi there

  pi = 3.14
##text = 'The value of pi is ' + pi      ## NO, does not work
text = 'The value of pi is ' + str(pi)  ## yes
```

For numbers, the standard operators, +, /, * work in the usual way. There is no ++ operator, but +=, -=, etc. work. If you want integer division, use 2 slashes -- e.g. 6 // 5 is 1

The "print" function normally prints out one or more python items followed by a newline. A "raw" string literal is prefixed by an 'r' and passes all the chars through without special treatment of backslashes, so r'x\nx' evaluates to the length-4 string 'x\nx'. "print" can take several arguments to change how it prints things out (see python.org print function definition) like setting "end" to "" to no longer print a newline after it finishes printing out all of the items.

```
raw = r'this\t\n and that'

# this\t\n and that
print(raw)

multi = """It was the best of times.
It was the worst of times."""

# It was the best of times.
#   It was the worst of times.
print(multi)
```

String Methods

Here are some of the most common string methods:

- s.lower(), s.upper() -- returns the lowercase or uppercase version of the string
- s.strip() -- returns a string with whitespace removed from the start and end

s.isalpha()/s.isdigit()/s.isspace()... -- tests if all the string chars are in the various character classes

s.startswith('other'), s.endswith('other') -- tests if the string starts or ends with the given other string

s.find('other') -- searches for the given other string (not a regular expression) within s, and returns the first index where it begins or -1 if not found

s.replace('old', 'new') -- returns a string where all occurrences of 'old' have been replaced by 'new'

s.split('delim') -- returns a list of substrings separated by the given delimiter. The delimiter is not a regular expression, it's just text. 'aaa,bbb,ccc'.split(',') -> ['aaa', 'bbb', 'ccc']. As a convenient special case s.split() (with no arguments) splits on all whitespace chars.

s.join(list) -- opposite of split(), joins the elements in the given list together using the string as the delimiter. e.g. '---'.join(['aaa', 'bbb', 'ccc']) -> aaa---bbb---ccc

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

Formatted string literals

You'll often see formatted string literals used in situations like:

```
    value = 2.791514
print(f'approximate value = {value:.2f}')  # approximate value = 2.79

car = {'tires':4, 'doors':2}
print(f'car = {car}') # car = {'tires': 4, 'doors': 2}
```

A formatted literal string is prefixed with 'f' (like the 'r' prefix used for raw strings). Any text outside of curly braces '{}' is printed out directly. Expressions contained in '{}' are are printed out using the format specification described in the format spec. There are lots of neat things you can do with the formatting including truncation and conversion to scientific notation and left/right/center alignment.

f-strings are very useful when you'd like to print out a table of objects and would like the columns representing different object attributes to be aligned like

```
address_book = [{'name':'N.X.', 'addr':'15 Jones St', 'bonus': 70},
 {'name':'J.P.', 'addr':'1005 5th St', 'bonus': 400},
 {'name':'A.A.', 'addr':'200001 Bdwy', 'bonus': 5},]

for person in address_book:
  print(f'{person["name"]:8} || {person["addr"]:20} ||
{person["bonus"]:>5}')

# N.X.     || 15 Jones St          ||    70
# J.P.     || 1005 5th St          ||   400
# A.A.     || 200001 Bdwy          ||     5
```

If Statement

Python does not use { } to enclose blocks of code for if/loops/function etc.. Instead, Python uses the colon (:) and indentation/whitespace to group statements. The boolean test for an if does not need to be in parenthesis (big difference from C++/Java), and it can have *elif* and *else* clauses (mnemonic: the word "elif" is the same length as the word "else").

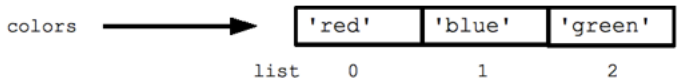
```
 if time_hour >= 0 and time_hour <= 24:
print('Suggesting a drink option...')
if mood == 'sleepy' and time_hour < 10:
```

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

Python Lists

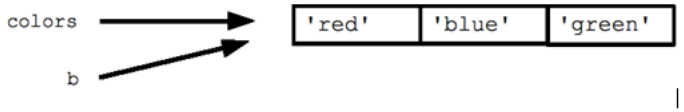
Python has a great built-in list type named "list". List literals are written within square brackets [ ]. Lists work similarly to strings -- use the len() function and square brackets [ ] to access data, with the first element at index 0. (See the official python.org list docs.)

```
colors = ['red', 'blue', 'green']
print(colors[0])    ## red
print(colors[2])    ## green
print(len(colors))  ## 3
```



Assignment with an = on lists does not make a copy. Instead, assignment makes the two variables point to the one list in memory.

```
b = colors   ## Does not copy the list
```



The "empty list" is just an empty pair of brackets [ ]. The '+' works to append two lists, so [1, 2] + [3, 4] yields [1, 2, 3, 4] (this is just like + with strings).

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

List Methods

Here are some other common list methods.

- list.append(elem) -- adds a single element to the end of the list. Common error: does not return the new list, just modifies the original.
- list.insert(index, elem) -- inserts the element at the given index, shifting elements to the right.
- list.extend(list2) adds the elements in list2 to the end of the list. Using + or += on a list is similar to using extend().
- list.index(elem) -- searches for the given element from the start of the list and returns its index. Throws a ValueError if the element does not appear (use "in" to check without a ValueError).
- list.remove(elem) -- searches for the first instance of the given element and removes it (throws ValueError if not present)

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

List Methods

- list.sort() -- sorts the list in place (does not return it). (The sorted() function shown later is preferred.)
- list.reverse() -- reverses the list in place (does not return it)
- list.pop(index) -- removes and returns the element at the given index. Returns the rightmost element if index is omitted (roughly the opposite of append()).

```
list = ['larry', 'curly', 'moe']
list.append('shemp')         ## append elem at end
list.insert(0, 'xxx')        ## insert elem at index 0
list.extend(['yyy', 'zzz'])  ## add list of elems at end
print(list)  ## ['xxx', 'larry', 'curly', 'moe', 'shemp', 'yyy', 'zzz']
print(list.index('curly'))    ## 2

list.remove('curly')         ## search and remove that element
list.pop(1)                  ## removes and returns 'larry'
print(list)  ## ['xxx', 'moe', 'shemp', 'yyy', 'zzz']
```

Common error: note that the above methods do not *return* the modified list, they just modify the original list.

```
list = [1, 2, 3]
print(list.append(4))    ## NO, does not work, append() returns None
## Correct pattern:
list.append(4)
print(list)  ## [1, 2, 3, 4]
```

List Build Up

One common pattern is to start a list as the empty list [], then use append() or extend() to add elements to it:

```
list = []          ## Start as the empty list
list.append('a')   ## Use append() to add elements
list.append('b')
```

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)

EUROPEAN NETWORK FOR 3D PRINTING OF BIOMIMETIC MECHATRONIC SYSTEMS - EMERALD

Summary

Python has become one of the most popular programming languages in the world in recent years. It's used in everything from machine learning to building websites and software testing. It can be used by developers and non-developers alike.

Python, one of the most popular programming languages in the world, has created everything from Netflix's recommendation algorithm to the software that controls self-driving cars. Python is a general-purpose language, which means it's designed to be used in a range of applications, including **data science**, **software and web development**, **automation**, and generally getting stuff done.

This results was realised with the EEA Financial Mechanism 2014-2021 financial support. Its content (text, photos, videos) does not reflect the official opinion of the Programme Operator, the National Contact Point and the Financial Mechanism Office. Responsibility for the information and views expressed therein lies entirely with the author(s)